SW 코딩 동아리

Algorithm DIJKSTRA

목차

1 알고리즘과 효율

2 Dynamic Programming

3 최단경로 알고리즘

4 Greedy Method

5 Dijkstra Algorithm

Part 1,
알고리즘과 효율

Part 1,

알고리즘과 효율

001 >>

알고리즘이란?

- 알고리즘은 주어진 문제를 논리적으로 해결하는 과정이다.
- 분석을 통해 작성한 알고리즘의 정확성을 파악 할 수 있고, 효율성을 정량적으로 나타낼 수 있다.
- 알고리즘은 프로그래밍 언어에 독립적이다.

---

002 >>

알고리즘의 효율성: 시간 복잡도(Time Complexity)

- 입력 크기(n)에 따라서 단위연산이 몇 번 수행되는지 결정하는 절차
- CPU 사용량에 비례

003 >>

알고리즘의 효율성: 공간 복잡도(Space Complexity)

- 입력 크기(n)에 따라서 연산을 수행하는데 필요한 자원을 결정하는 절차
- RAM 사용량에 비례

Part 2,
Dynamic Programming

Dynamic Programming

001 >> 동적 프로그래밍(동적 계획법)

- 문제해결을 위해 문제를 여러 개의 하위 문제로 나누어 해결한 뒤 결합하여 해결하는 방법
- 각 하위 문제에서 얻은 결과를 저장, 이후에 문제를 해결하는 자료로 재사용

002 >> 막대 자르기 문제

- 막대 길이별로 가격이 상이
- 막대 길이(n)에 대하여 최대의 이익 R_n 산출

길이	1	2	3	4	5	6	7	8	9	10
가격	1	5	8	9	10	17	17	20	24	30

Part 2,

Dynamic Programming

길이	1	2	3	4	5	6	7	8	9	10
가격	1	5	8	9	10	17	17	20	24	30

(0) Max = 0

(1) 1개로 나눌 경우

Max = 10 > 0

(2) 2개로 나눌 경우

Max = 10 ≥ 1 + 9

Max = 5 + 8 > 10

Max = 13 ≥ 8 + 5

Max = 13 ≥ 9 + 1

⋮

(n) n개로 나눌 경우

Max = 13 ≥ 1 + 1 + 1 + 1 + 1

Dynamic Programming

001 >> 시간 복잡도(Time Complexity)

- n = 5: ${}_4C_0 + {}_4C_1 + {}_4C_2 + {}_4C_3 + {}_4C_4 = 32$
- n: ${}_nC_0 + {}_nC_1 + \ldots + {}_nC_n = 2^n$
- $O(2^n)$

---

002 >> 공간 복잡도(Space Complexity)

- 사용한 변수 Max 1개
- O(1)

Part 2,

Dynamic Programming

길이	1	2	3	4	5	6	7	8	9	10
가격	1	5	8	9	10	17	17	20	24	30

(1) n = 1

Max = 1

(2) n = 2

Max = 5

Max = 5 > 1 + 1

(3) n = 3

Max = 8

Max = 8 > 1 + 5

.
.
.

(n) n개로 나눌 경우 Max = $\max(R_i + R_{n-i}) = \max(P_i + R_{n-i})$ (단, $1 \le i \le 10$)

Dynamic Programming

001 >> 시간 복잡도(Time Complexity)

- $R_0 \sim R_{n-1}$ 을 모를 경우

  n = 5: 1 + 2 + 3 + 4 + 5 = 32

  n: 10(n - 10) + 55 = 10n – 45 (단, n ≥ 10)

  O(n)

- $R_0 \sim R_{n-1}$ 을 알 경우

  n = 5: 5

  n: 10 (단, n ≥ 10)

  O(1)

002 >> 공간 복잡도(Space Complexity)

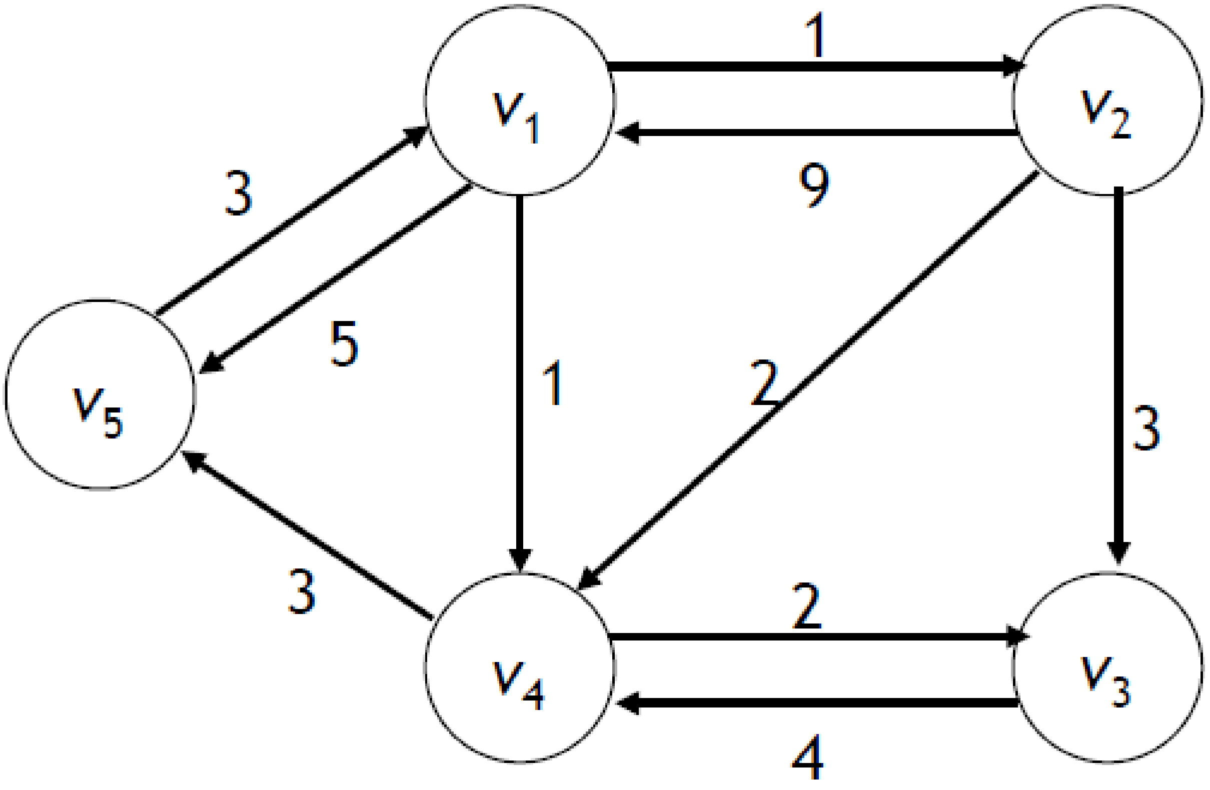
- $R_0 \sim R_{n-1}$, Max 변수 총 (n + 1)개 변수 사용
- O(n)

Part 3,

최단경로 알고리즘

001 >> 최단경로 문제

- 한 도시에서 다른 도시로 가장 빨리 갈 수 있는 항로를 찾는 문제



최단경로 알고리즘

001 >> 무작정 알고리즘 (Brute-force Algorithm)

- 한 노드에서 다른 노드로의 모든 가능한 경로의 길이를 구한다.
- 그 경로들 중에서 최소 길이를 구한다.
- O(n!)

002 >> Dynamic Programming 기반 최단경로: Floyd Algorithm

- W[i][j]는 그래프의 연결상태를 표시
- k이하의 노드를 거쳐 i노드부터 j노드까지 가는 최단경로비용: $D^k[i][j]$

$W[i][j]$	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

$$W[i][j] = \begin{cases} \text{가중치} & v_i\text{에서 } v_j\text{로 가는 이음선이 있는 경우} \\ \infty & v_i\text{에서 } v_j\text{로 가는 이음선이 없는 경우} \\ 0 & i = j \text{ 인 경우} \end{cases}$$

최단경로 알고리즘

$$D^{(k)}[i][j]=\min(\underbrace{D^{(k-1)}[i][j]}_{\text{경우 1}},\underbrace{D^{(k-1)}[i][k]+D^{(k-1)}[k][j]}_{\text{경우 2}})$$

$W[i][j]$	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

$D[i][j]$	1	2	3	4	5
1	0	1	3	1	4
2	9	0	3	2	5
3	∞	∞	0	4	7
4	6	∞	2	0	3
5	3	4	∞	4	0

$D[i][j]$	1	2	3	4	5
1	0	1	3	1	4
2	8	0	3	2	5
3	10	11	0	4	7
4	6	7	2	0	3
5	3	4	6	4	0

Practice 1

최단경로 알고리즘

001 >> 시간 복잡도(Time Complexity)

- for (int i = 0; i < n; i++) 3개 사용: n^3
- O(n^3)

002 >> 공간 복잡도(Space Complexity)

- D[n][n] 배열 사용: n^2
- O(n^2)

Part 4,

Greedy Method

Greedy Algorithm

001 >> 탐욕적 알고리즘(Greedy Algorithm)

- 결정 순간마다 그때 최적인 해답으로 선택함으로써 최종적인 해답에 도달
- 각 하위 문제에서 얻은 결과를 저장, 이후에 문제를 해결하는 자료로 재사용

---

002 >> 탐욕적 알고리즘 설계 절차

- 선정과정(selection procedure)

  현재상태에서 가장 좋다고 생각되는 해답(greedy)을 해답모음(solution set)에 포함

- 적정성점검(feasibility check)

  새로 얻은 해답모음이 적절한지를 결정

- 해답점검(solution check)

  새로 얻은 해답모음이 최적의 해인지를 결정

Greedy Algorithm

001 >> 최소 동전수로 거스름돈을 주는 문제

- 거스름 돈 x
- 가치가 높은 동전부터 x가 초과되지 않도록 계속 준다
- 이 과정을 가치가 높은 동전부터 내림순으로 총액이 x가 될 때까지 반복

002 >> 최소 동전수로 거스름돈을 주는 문제 적용

- x = 16

- 선정과정(selection procedure) / 적정성점검(feasibility check)
  (1) 10원, 5원, 1원: 10원×1개, 5원×1개, 1원×1개
  (2) 12원, 10원, 5원, 1원: 12원×1개, 1원×4개

- 해답점검(solution check)
  (1) 10원, 5원, 1원: 3개 → Optimal(최적)
  (2) 12원, 10원, 5원, 1원: 5개 → Not Optimal(최적 아님)

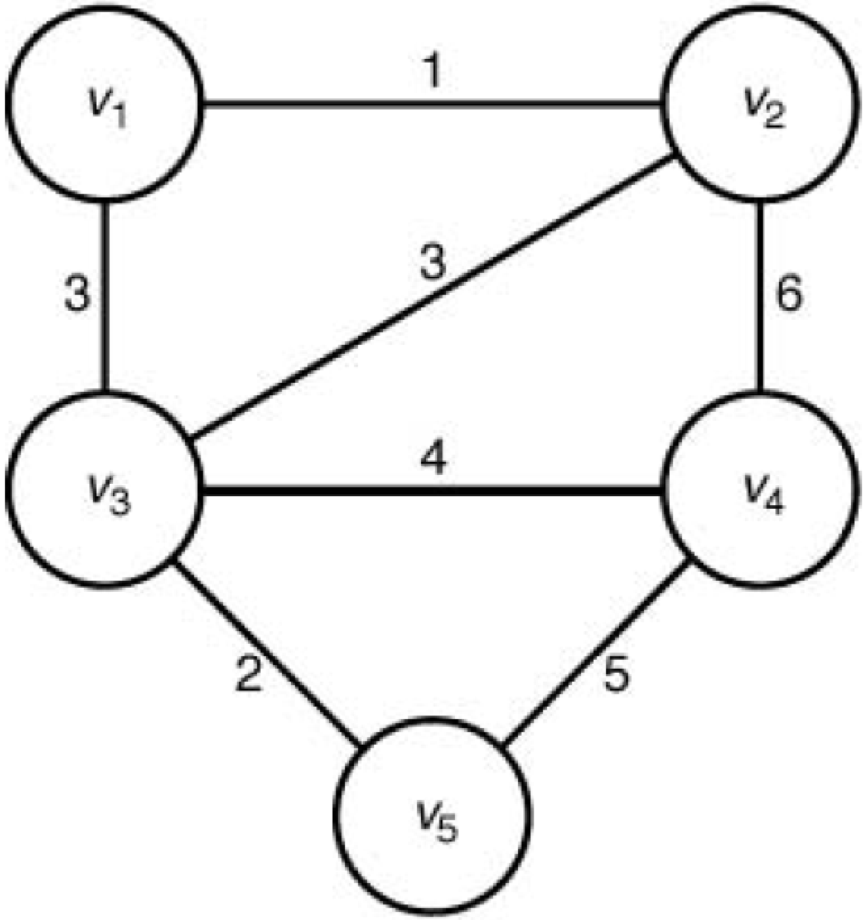
Part 5,

Dijkstra Algorithm

Dijkstra Algorithm

001 >> 최단경로 문제

- 한 도시에서 다른 도시로 가장 빨리 갈 수 있는 항로를 찾는 문제



Dijkstra Algorithm

001 >> 최단경로 문제

- F: = 0;
- Y: = {v1};
- 최종해답을 얻지 못하는 동안 다음 절차를 계속 반복

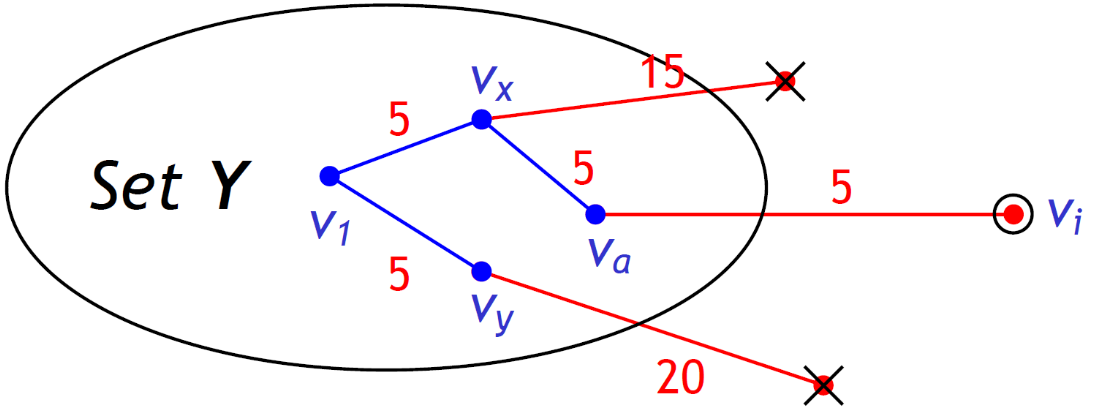
(a) 선정절차 / 적정성점검

V - Y의 정점 중, v1에서 Y에 속한 정점만을 거치는 최단 경로가 되는 정점 v를 선정

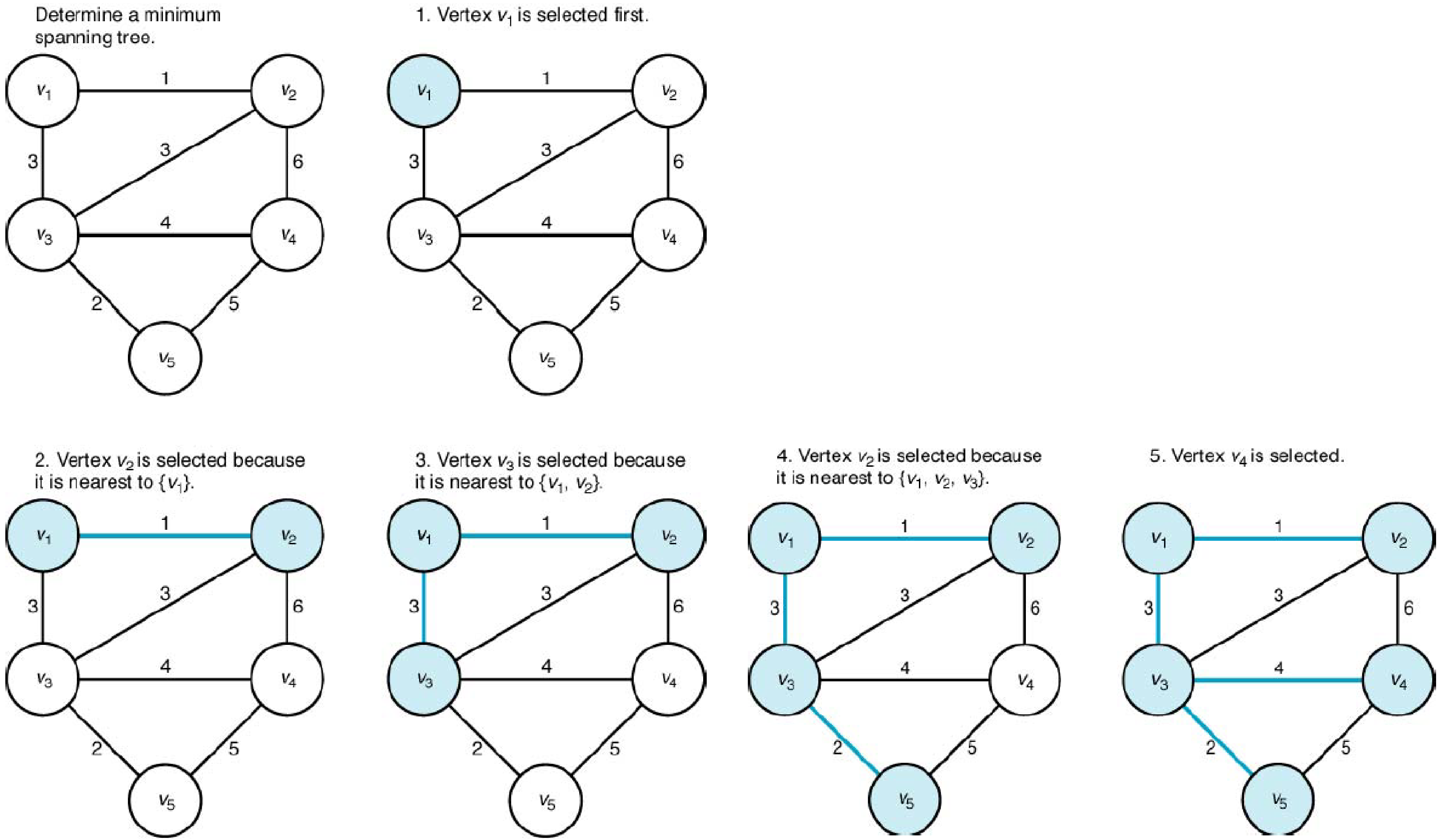
(b) 그 정점 v를 Y에 추가 (아래 그림에서 vi)

(c) v에서 F로 이어지는 최단경로상의 이음선을 F에 추가

(d) 해답 점검: Y = V가 되면, T = (V, F)가 최단경로를 나타내는 그래프



Dijkstra Algorithm



Dijkstra Algorithm

001 >> 최단경로 문제

- F: = 0;
- Y: = {v1};
- 최종해답을 얻지 못하는 동안 다음 절차를 계속 반복

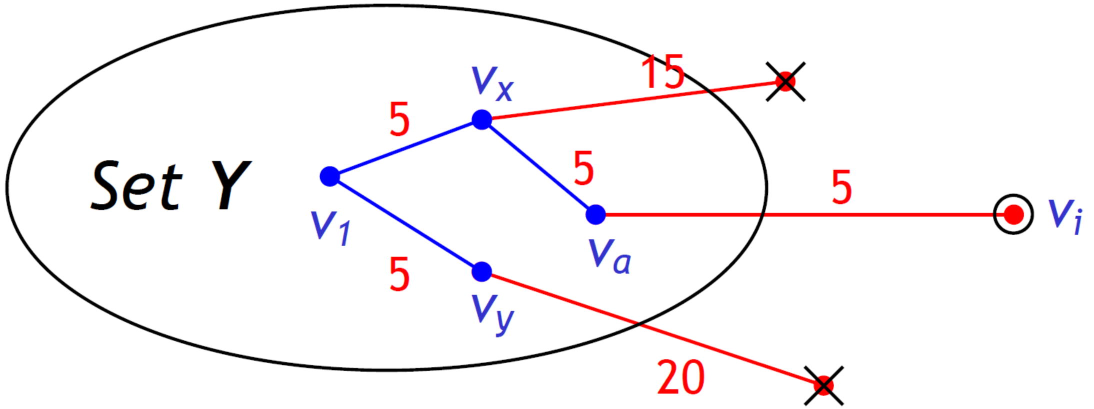
(a) 선정절차 / 적정성점검

V - Y의 정점 중, v1에서 Y에 속한 정점만을 거치는 최단 경로가 되는 정점 v를 선정

(b) 그 정점 v를 Y에 추가 (아래 그림에서 vi)

(c) v에서 F로 이어지는 최단경로상의 이음선을 F에 추가

(d) 해답 점검: Y = V가 되면, T = (V, F)가 최단경로를 나타내는 그래프



Practice 2

최단경로 알고리즘

001 >> 시간 복잡도(Time Complexity)

- for (int i = 0; i < n; i++) 2개 사용: n^2
- O(n^2)
- Heap을 통해 구현하면 O($\mathrm{mlog}\, n$)
- Fibonacci Heap을 통해 구현하면 O($\mathrm{m} + \mathrm{nlog}\, n$)

002 >> 공간 복잡도(Space Complexity)

- D[n][n] 배열 사용: n^2
- O(n^2)

Complete Code

- Algorithm Runtime Comparison -

Q&A